



CrowdStrike

ACPI 5.0 Rootkit Attacks “Againts” Windows 8

Alex Ionescu
Chief Architect

SyScan 2012

@aionescu
alex@crowdstrike.com

Bio

- Reverse engineered Windows kernel since 1999
 - Previously lead kernel developer for ReactOS Project
- Interned at Apple for a few years (Core Platform Team)
- Co-author of Windows Internals 5th and 6th Edition
 - Also instructor and contributor to Windows Internals seminar for David Solomon Expert seminars
- Founded Winsider Seminars & Solutions Inc., to provide services and Windows Internals training for enterprise/government
- Now Chief Architect at CrowdStrike
 - Security startup focused on attribution, received \$26M in funding

Introduction

Outline

- Introduction
- Microsoft Vendor-Specific ACPI Tables
 - CSRT and HAL Extensions
 - WPBT and Platform Binaries
 - WDAT and Watchdog Timers
- Abusing ACPI for Fun & Profit
- Software ACPI Emulation in Windows
- Conclusion

What This Talk Is (Not) About

- A rootkit/malware persistence technique
- Hardware attacks from software
- **Require privileged user token -- not a 'security vulnerability'**

Also, lost my laptop+slides+demos on the plane a few days ago, so I'm sorry for the last minute deck

Recommended Reading

- ACPI 5.0 Specification (<http://www.acpi.info>)
- CSRT and WPBT Specification (<http://msdn.microsoft.com/en-us/library/windows/hardware/gg463220.aspx> -- Dev Center - Hardware > Learn > Systems > Power Management and ACPI -> Architecture and Driver Support -> Microsoft-defined ACPI Table Specifications)
- WDAT Specification (<http://msdn.microsoft.com/en-us/library/windows/hardware/gg463320.aspx> -- Dev Center - Hardware > Learn > Systems > System Internals > Hardware Watchdog Timers Design Specification)

Microsoft Vendor-Specific ACPI Tables

Core System Resource Table (CSRT)

- New to ACPI 5.0 and supported in Windows 8
- Specifies Interrupt Controller, Timer, or DMA Controller
- Defines VID and PID
 - SVID and SPID optional
- Defines UID for this particular piece of silicon
- Defines custom data for the hardware
- CSRT is an array of resource groups
 - Each resource group has a header, custom data, and an array of resource descriptors
- Each resource descriptor has a header and custom data

HAL Extensions

- New mechanism added to Windows 8 to support ARM SoCs
- Instead of multiple vendor HALs, one ACPI HAL that loads custom 3rd party DLLs based on VID/PID of detected devices
- Custom-loaded by HAL, only certain APIs allowed:

```
.data:80042288 _HalpExtensionImports dd offset _HalpExtRegisterResourceDescriptor@16 |
.data:80042288 ; DATA XREF: HalpExtInitExtensions(x)+B2↓o
.data:80042288 ; HalpExtRegisterResourceDescriptor(x,x,x,x)
.data:8004228C dd offset _HalpExtGetNextResourceDescriptor@24 ; HalpExtGetNextResourceDescriptor(x,x,x,x,x,x)
.data:80042290 dd offset _HalpExtGetAcpiTable@16 ; HalpExtGetAcpiTable(x,x,x,x)
.data:80042294 dd offset _HalMapIoSpace@16 ; HalMapIoSpace(x,x,x,x)
.data:80042298 dd offset _HalUnmapIoSpace@8 ; HalUnmapIoSpace(x,x)
.data:8004229C dd offset _HalRegisterPermanentAddressUsage@12 ; HalRegisterPermanentAddressUsage(x,x,x)
.data:800422A0 dd offset _HalSetTimerProblem@12 ; HalSetTimerProblem(x,x,x)
.data:800422A4 dd offset _HalUpdateTimerCapabilities@12 ; HalUpdateTimerCapabilities(x,x,x)
.data:800422A8 dd offset _RtlRaiseException@4 ; RtlRaiseException(x)
.data:800422AC dd offset _HalpExtBugCheckEx@20 ; HalpExtBugCheckEx(x,x,x,x,x)
```

- Windows Boot Loader parses registry entries in CurrentControlSet\Control\HAL and loads each DLL specified
- HAL queries CSRT and compares VID/PID with DLL descriptors
- !halext in Windbg allows you to see loaded HAL Extensions

Windows Platform Binary Table (WPBT)

- New to ACPI 5.0 and supported in Windows 8
- Specifies a physical address containing a PE binary
- Describes its layout
 - Only one layout supported: Flat PE without expanded sections and no relocations applied
- Describes its type
 - Only one type supported: Native NT Binary launched by Smss.exe
- Also defines command-line argument length and buffer (in UNICODE_STRING format)
- Disabled in Safe Mode and WinRE
- OS Executes _PBS method in DSDT (AML code) to notify BIOS of execution state and parameters

WPBT Implementation

- During user-mode startup, Smss calls *NtQuerySystemInformation* with *SystemPlatformBinaryInformation*
 - Kernel calls *ExpGetSystemPlatformBinary* which parses the WPBT
 - Kernel uses *MmMapIoSpace* to map the binary in virtual address space (kernel buffer)
 - Kernel makes a copy into the buffer supplied by Smss
 - Smss receives `SYSTEM_PLATFORM_BINARY_INFORMATION` structure with size and buffer filled out
- SMSS then writes the file to disk and launches it
 - `\Windows\System32\Wpbbin.exe`
- Once SMSS initializes registry (*NtInitializeRegistry*), table is dropped and cannot be queried again
- Image must be signed (`/INTEGRITYCHECK`) but not page hashed

Watch Dog Action Table (WDAT)

- Refined version of the Watch Dog Resource Table (WDRT)
 - New to Windows Vista
- Defines a series of “Watchdog Instruction Entries”
- Each entry defines an action that the OS may want to take
 - Reset/Init/Query/Set of State/Countdown/Status
- Then defines the register associated with that action
 - Follows ACPI GEN_ADDR Spec (5.2.3.1 Address Space Format)
 - ASID (PCI Space, RAM, I/O Space)
 - Access Size (Byte, Word, Dword, QWord)
 - 64-bit Address
- Finally, describes action to take on the registry
 - *Read and compare value or read and store value*
 - *Write value or Write stored value*

HAL Management of ACPI Tables

- ACPI Tables not really owned by ACPI.SYS
 - HAL has well-known HAL_DISPATCH and HAL_PRIVATE_DISPATCH, but also undocumented/not-well-known HAL_PM_DISPATCH and HAL_ACPI_DISPATCH

```
lkd> dps hal!HalAcpiDispatchTable
fffff800`03c254c0 00000004`48414c20
fffff800`03c254c8 fffff800`03c115d0 hal!HaliAcpiTimerCarry
fffff800`03c254d0 fffff800`03c39bf8 hal!HaliAcpiMachineStateInit
fffff800`03c254d8 fffff800`03c39e7c hal!HaliAcpiQueryFlags
fffff800`03c254e0 fffff800`03c34fc8 hal!HalpAcpiPicStateIntact
fffff800`03c254e8 fffff800`03c34dd8 hal!HalpRestoreInterruptControllerState
fffff800`03c254f0 fffff800`03c0c7dc hal!HaliPciInterfaceReadConfig
fffff800`03c254f8 fffff800`03c0c87c hal!HaliPciInterfaceWriteConfig
fffff800`03c25500 fffff800`03c12fb0 hal!HalpGetApicVersion
fffff800`03c25508 fffff800`03c0c7c4 hal!HaliSetMaxLegacyPciBusNumber
fffff800`03c25510 fffff800`03c3aa50 hal!HaliIsVectorValid
fffff800`03c25518 fffff800`03c07ba8 hal!HalAcpiGetTableDispatch
fffff800`03c25520 fffff800`03c07c04 hal!HalAcpiGetRsdpDispatch
fffff800`03c25528 fffff800`03c07c14 hal!HalAcpiGetFacsMappingDispatch
fffff800`03c25530 fffff800`03c07c24 hal!HalAcpiGetAllTablesDispatch
```

- hal!HalAcpiGetTableDispatch actually owns querying the tables
- HAL Caches the tables on start-up to avoid repeated lookups in RSDT
- See halinit.c in ReactOS ACPI HAL

!acpicache

- WinDBG extension that shows tables cached by the HAL
 - Loops `hal!HalpAcpiTableCacheList`
- Fails to work since `LIST_ENTRY` is not defined
 - Public symbols do not have “typedefs”, `_LIST_ENTRY` is real name
- Even if `LIST_ENTRY` is present, depends on undocumented type
 - `ACPI_CACHED_TABLE`
- Little known fact about PDB files: they are streams
 - Microsoft C compiler (CL.EXE) can append data to an existing symbol file
 - Does not break the hash!
 - `CL /Zi /Gz /c /Fd<symbolfile> <dummyfile.c>`
- DEMO!

Software ACPI Emulation in Windows

ACPI Table Override Mechanism

- Implemented in Windows 2000
 - Well-known dirty secret of OEMs, but never published/discovered externally
- Windows Boot Loader reads special ACPI Table file from disk

```
.text:0000000000402A02      mov     rax, [rdi+8]
.text:0000000000402A06      lea     rdx, aSystem32Acpita ; "system32\\acpitabl.dat"
.text:0000000000402A0D      mov     rcx, rdi          ; Destination
.text:0000000000402A10      mov     [rax+r12*2], r15w
.text:0000000000402A15      mov     [rdi], bp
.text:0000000000402A18      call    BlAppendUnicodeToString
```

- Populated in AcpiTable field of LOADER_PARAMETER_EXTENSION

```
.text:0000000000402A70      mov     r9d, [rsp+88h+arg_8]
.text:0000000000402A78      mov     r8, [rsp+88h+var_38]
.text:0000000000402A7D      lea     rdx, [rax+78h] ; AcpiTable
.text:0000000000402A81      lea     rcx, [rax+70h] ; AcpiTableSize
.text:0000000000402A85      call    OslMergeAcpiTables
```

- HAL Reads in HalpAcpiTableCacheInit and adds to its cache

```
00754      /* Check for compatible loader block extension */
00755      if (LoaderExtension && (LoaderExtension->Size >= 0x58))
00756      {
00757          /* Compatible loader: did it provide an ACPI table override? */
00758          if ((LoaderExtension->AcpiTable) && (LoaderExtension->AcpiTableSize))
00759          {
00760              /* Great, because we don't support it! */
00761              DPRINT1("ACPI Table Overrides Not Supported!\n");
```


The Evils of ACPITABL.DAT

- You can side-load your own DSDT
 - Run custom AML rootkits without having to flash the BIOS (John Heasman, BlackHat '06 & others)
 - You would have to re-create the original OEM DSDT, however
- You can side-load your own WPBT
 - Run custom user-mode binaries from within the table itself, with SYSTEM privileges
 - You would need to control RAM, however, as the command-line and image pointer are loaded by a RAM address
- You can side-load your own CSRT
 - Now HAL will try loading a custom HAL Extensions
 - You would still have to drop the HAL extension on the system and have the registry keys correctly setup

Abusing the WDAT

- Or you could side-load your custom WDAT
 - Covert: not obvious as to what is happening
 - No other code/registry keys required on the system
- At the end of the day, this is what an evil ACPITABL.DAT looks like

0000h:	B7 44 41 54 4C 01 00 00 01 3D 41 6C 65 78 20 20	WDATL....=Alex
0010h:	49 6F 6E 65 73 63 75 20 4F 77 6E 73 54 68 69 73	Ionescu OwnsThis
0020h:	42 69 6F 73 20 00 00 00 FF 00 FF FF FF 00 00 00	Bios ...ÿ.ÿÿÿ...
0030h:	58 02 00 00 FF 03 00 00 02 00 00 00 81 00 00 00	X...ÿ.....
0040h:	0B 00 00 00 01 02 00 00 00 20 00 03 14 38 29 00	8).
0050h:	00 00 00 00 00 00 00 00 FF FF FF FF 01 02 00 00	ÿÿÿÿ....
0060h:	00 20 00 03 00 4A 10 00 00 00 00 00 FF 05 E0 02	J.....ÿ.à.
0070h:	FF FF FF FF 01 02 00 00 00 20 00 03 04 4A 10 00	ÿÿÿÿ..... ...J..
0080h:	00 00 00 00 DF FF 0F 31 FF FF FF FF 01 02 00 00	Bÿ.1ÿÿÿÿ....
0090h:	00 20 00 03 08 4A 10 00 00 00 00 00 C2 04 00 00	J.....Â...
00A0h:	FF FF FF FF 01 02 00 00 00 20 00 03 50 40 10 00	ÿÿÿÿ..... ..P@..
00B0h:	00 00 00 00 00 2A D0 FF FF FF FF FF 08 02 00 00	*Bÿÿÿÿÿ....
00C0h:	00 20 00 03 30 63 10 00 00 00 00 00 41 41 41 41	. ..0c.....AAAA
00D0h:	FF FF FF FF 09 02 00 00 00 20 00 03 30 63 10 00	ÿÿÿÿ..... ..0c..
00E0h:	00 00 00 00 41 41 41 41 FF FF FF FF 0A 02 00 00	AAAAÿÿÿÿ....
00F0h:	00 20 00 03 30 63 10 00 00 00 00 00 41 41 41 41	. ..0c.....AAAA
0100h:	FF FF FF FF 0B 02 00 00 00 20 00 03 30 63 10 00	ÿÿÿÿ..... ..0c..
0110h:	00 00 00 00 41 41 41 41 FF FF FF FF 20 02 00 00	AAAAÿÿÿÿ ...
0120h:	00 20 00 03 30 63 10 00 00 00 00 00 41 41 41 41	. ..0c.....AAAA
0130h:	FF FF FF FF 21 02 00 00 00 20 00 03 30 63 10 00	ÿÿÿÿ!..... ..0c..
0140h:	00 00 00 00 41 41 41 41 FF FF FF FF	AAAAÿÿÿÿ

Crafting an ACPITABL.DAT

- Set signature to WDAT and fill out DESCRIPTION_HEAER according to ACPI Spec (5.2.6 System Description Table Header)
- Remember to set the checksum byte
 - 8-bit sum of entire table (including checksum byte) should yield zero
- Set the table size
- Any data after the table you've defined is treated as another table
- No maximum size to ACPITABL.DAT
- CAREFUL:
 - Corrupted ACPITABL.DAT can lead to unbootable system
 - If extremely unlucky, badly formed ACPITABL.DAT can lead to I/O and/or RAM access that could brick your machine
 - Ie: Flashing your BIOS with 0x41414141 ☺

Abusing ACPI for Fun & Profit

WDAT(TACK)

- A WDAT could be written in a very complex way such that the read-and-compare as well as the read-and-store + write-stored-value perform some sort of attack on the system
- However, physical RAM addresses must be used and either discovered (complex) or known in advance
- Modification to RAM must be stable across boots, and preferably across OS releases
- Would allow malware to simply drop ACPITABL.DAT on disk (not checked by any known AV/IDS/HIPS) and wait for reboot to ensure persistence
 - RAM modification could be to add code to the system, patch code, or even DKOM hooks/hiding from process list, etc

HAL Heap

- Fixed region of kernel memory since NT 4
 - See “Bypassing Windows 7 Kernel ASLR” by Stefan Le Berre
- Starts at 0xFFD00000
 - See hal!HalpHeapStart, hal!HalpHeapEnd
- Executable memory up until Windows 8 Consumer Preview
 - Sometimes contains code -- have seen pages executable on Windows 8 CP as well, but could not repro on all machines
- Used by HAL to store ACPI Cached Tables
- Undocumented WinDBG extension shows all PTEs
 - !halpte
 - Note distribution of physical RAM pages to virtual addresses – not random!

HAL Changes in Windows 8

- Due to ARM SoC support, major HAL rewrite in Windows 8
- Now have Timer Objects, Interrupt Controller Objects, DMA Controller Objects...
 - Each is linked together, has its own “class” (still in C) as well as function table
 - See `dt hal!_INTERRUPT_FUNCTION_TABLE` for example
- `hal!HalpRegisteredTimers`, `hal!HalpRegisteredPics`, etc...
 - Underlying objects are allocated from HAL Heap!
- Deterministic allocator algorithm and PFN selection algorithm == deterministic function tables
 - Can achieve execution by overwriting a commonly executed callback
 - `QueryValueCallback` (Timer) or `RequestInterruptCallback` (IC)

Bypassing NX on Windows 8 HAL Heap

- On Windows 7, HAL Heap is executable, so can simply drop the code there
 - Note, however, that there are no easy function tables to hook – execution must be achieved through other means
- NX is 63-bit on PTE (0x8000000000000000)
- For optimization, Windows has a flat array of all PTEs valid for the current process at 0xC0000000 (x86 – different address on x64)
 - ie: Hardcoded virtual address for each PTE in HAL Heap range
- Forensic guru's little dirty secret:
 - Windows allocates System Page Directory (CR3) at fixed physical location!
 - Fixed/known page tables for certain ranges too (such as HAL Heap PDE)
- We can just edit the PTE as part of the attack!

Watchdog Action Attack Table

- Simple proof of concept attack in 3 steps
 - Edit Hal Heap PTE to make it executable
 - Write rootkit code
 - Patch HAL Timer Query Callback for the TSC (first timer on the system)
- WATCHDOG_INSTRUCTION_WRITE_VALUE
 - 32-bit write (0x00000000) @ HAL Heap PTE + 4 (clears NX Bit)
- WATCHDOG_INSTRUCTION_WRITE_VALUE
 - 32-bit write (inc dword ptr ds:[0xffdf02E0]) @ HAL Heap
 - 32-bit write (rdtsc) @ HAL Heap
 - 32-bit write (ret 4) @ HAL Heap
- WATCHDOG_INSTRUCTION_WRITE_VALUE
 - 32-bit write (0xFFD0YYYY) @ HAL Heap TSC Timer Function Table

Other Possibilities

- Since System CR3 is well-known, can edit PTEs and make certain pages Ring 3 for user-mode backdoor
- Can patch code in kernel memory to insert backdoor/rootkit
 - Similar to snare's EFI presentation on OS X
- KPCR and KPRCB for CPU 0 are at well-known location
 - 0xFFDF0000
 - KPRCB contains function pointers for Idle CPU Callback and C-state transitions at well-known offset
 - Read of well-known PTE offset leads to discovery of KPCR physical page
 - Overwriting Idle CPU Callback leads to pwnage
- Other than RAM access, watchdog timer also has I/O access
 - Could use I/O ports to flash ROM, or other mayhem

DEMO

Conclusion

Key Takeaways

- Other than AML attacks which have been published in the past, operating systems are also vulnerable to table-based attacks
 - On-going Linux work to support WDAT
- ACPI Spec does not prohibit GEN_ADDR structures that map to OS-owned memory
- Windows 8 has new ASEP mechanisms leveraging ACPI 5.0
 - Windows Platform Binary Table allows loading PE file from RAM
 - Core System Resource Table allows loading HAL Extension
- Windows 2000 and later support definition of ACPI Tables on disk
 - Can be combined with ASEP and Table-based attacks for covert code execution/rootkit persistence as part
 - Bypasses protection afforded by Signed BIOS/UEFI

Defense-in-depth Suggestions

- WPBT binaries should have a special signature with a custom Publisher, and only signed by Microsoft (similar to ELAM drivers)
 - The same should be done for HAL Extensions
- ACPITABL.DAT should not allow certain kinds of tables to be located in it
 - If ACPITABL.DAT is used, an Event Log or other visible security notification should be generated by the kernel
 - Group policy setting should allow Administrators to disable ACPITABL.DAT mechanism, much like Shim Engine, SwitchBack, and other AppCompat features
- Kernel should not blindly use *MmMapIoSpace* on physical addresses from ACPI
 - WDAT and other similar tables should only contain MMIO pointers

Checking your ACPI Tables

- Windows API provides *EnumSystemFirmwareTables* and *GetSystemFirmwareTable*
 - Have written small tool to check for presence of WDAT and validate (WIP)
 - My new laptop actually has a legitimate WDAT!

```
C:\Users\Alex.Ionescu\Documents\Visual Studio 11\Projects\GetAcpiTables\Debug\GetAcpiTables.exe
ACPI Table Found: SSDT
Watchdog Entries: 20
Watchdog PCI Device: ff.ff.ff
Watchdog Timer Period: 6000 (min: 2 max: 1023)
Entry 0 Action: 1 (with flags 0x2) at address 0x460 (Value: 0x0 Mask: 0x3ff)
Entry 1 Action: 4 (with flags 0x1) at address 0x460 (Value: 0x0 Mask: 0x3ff)
Entry 2 Action: 5 (with flags 0x1) at address 0x472 (Value: 0x0 Mask: 0x3ff)
Entry 3 Action: 6 (with flags 0x83) at address 0x472 (Value: 0x0 Mask: 0x3ff)
Entry 4 Action: 8 (with flags 0x0) at address 0x468 (Value: 0x0 Mask: 0x1)
Entry 5 Action: 9 (with flags 0x82) at address 0x468 (Value: 0x0 Mask: 0x800)
Entry 6 Action: 9 (with flags 0x2) at address 0x70 (Value: 0x45 Mask: 0xff)
Entry 7 Action: 9 (with flags 0x82) at address 0x71 (Value: 0x1 Mask: 0x1)
Entry 8 Action: a (with flags 0x0) at address 0x468 (Value: 0x1 Mask: 0x1)
Entry 9 Action: b (with flags 0x82) at address 0x468 (Value: 0x800 Mask: 0x800)
Entry 10 Action: b (with flags 0x2) at address 0x70 (Value: 0x45 Mask: 0xff)
Entry 11 Action: b (with flags 0x82) at address 0x71 (Value: 0x0 Mask: 0x1)
Entry 12 Action: 10 (with flags 0x0) at address 0x46a (Value: 0x0 Mask: 0x3)
Entry 13 Action: 11 (with flags 0x82) at address 0x46a (Value: 0x0 Mask: 0x30)
Entry 14 Action: 12 (with flags 0x0) at address 0x46a (Value: 0x1 Mask: 0x3)
Entry 15 Action: 13 (with flags 0x82) at address 0x46a (Value: 0x10 Mask: 0x30)
Entry 16 Action: 20 (with flags 0x2) at address 0x70 (Value: 0x45 Mask: 0xff)
Entry 17 Action: 20 (with flags 0x0) at address 0x71 (Value: 0x1 Mask: 0x1)
Entry 18 Action: 21 (with flags 0x2) at address 0x70 (Value: 0x45 Mask: 0xff)
Entry 19 Action: 21 (with flags 0x82) at address 0x71 (Value: 0x0 Mask: 0x1)
```

- You can also fix your symbols and use !acpicache in Windbg

QA

- Greetz/shouts to: Matthieu Suiche, Tarjei Mandt, Bruce Dang, Loukas (snare)
- Stay tuned for more Windows 8 security talks this year!

PS. Asiana Airlines: Can I has ACPI rootkit laptop back? kthxbai.



crowdStrike